IN THE SPECIFICATION

Please amend the paragraph beginning on page 6, line 10 as follows:

HTTP: Short for *HyperText Transfer Protocol*, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when a user enters a URL in his or her browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page. Further description of HTTP is available in *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1*, the disclosure of which is hereby incorporated by reference. *RFC 2616* is available from the World Wide Web Consortium (W3)~~, and is currently available via the Internet at *http://www.w3.org/Protocols/*~~.

Please amend the paragraph beginning on page 6, line 35 as follows:

XML: Short for *Extensible Markup Language*, a specification developed by the W3C. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations. For further description of XML, see, e.g., *Extensible Markup Language (XML) 1.0* specification which is available from the World Wide Web Consortium ~~(www.w3.org)~~, the disclosure of which is hereby incorporated by reference. ~~The specification is also currently available on the Internet at *http://www.w3.org/TR/REC-xml.*~~

Please amend the paragraph beginning on page 13, line 20 as follows:

Illustrated in Fig. 3, a computer software system 300 is provided for directing the operation of the computer system 200. Software system 300, which is stored in system memory (RAM) 202 and on fixed storage (e.g., hard disk) 216, includes a kernel or operating system (OS) 310. The OS 310 manages low-level aspects of computer operation, including managing execution of processes, memory allocation, file input and output (I/O), and device I/O. One or more application programs, such as client application software or "programs" 301 (e.g., 301a, 301b, 301c, 301d) may be "loaded" (i.e., transferred from fixed storage ~~116~~ 216 into memory 202 ~~102~~) for execution by the system 200 ~~100~~.

Please amend the paragraph beginning on page 13, line 28 as follows:

System 300 includes a graphical user interface (GUI) 315, for receiving user commands and data in a graphical (e.g., "point-and-click") fashion. These inputs, in turn, may be acted upon by the system ~~100~~ 200 in accordance with instructions from operating system 310, and/or client application module(s) 301. The GUI 315 also serves to display the results of operation from the OS 310 and application(s) 301, whereupon the user may supply additional inputs or terminate the session. Typically, the OS 310 operates in conjunction with device drivers 320 (e.g., "Winsock" driver -- Windows' implementation of a TCP/IP stack) and the system BIOS microcode 330 (i.e., ROM-based microcode), particularly when interfacing with peripheral devices. OS 310 can be provided by a conventional operating system, such as Microsoft® Windows 9x, Microsoft® Windows NT, Microsoft® Windows 3000, or Microsoft® Windows XP, all available from Microsoft Corporation of Redmond, WA. Alternatively, OS 310 can also be an alternative operating system, such as the previously- mentioned operating systems.

Please amend the paragraph beginning on page 17, line 6 as follows:

In order to represent the products a fulfiller can supply, a bit vector is used, as follows:

F_1.bv[101] (which is $\underline{P_1}$ ~~P1~~and $\underline{P_3}$ ~~P3~~)
F_2.bv[001] (which is $\underline{P_3}$ ~~P3~~)
F_3.bv[111] (which is $\underline{P_1}$ ~~P1~~, $\underline{P_2}$ ~~P2~~ and $\underline{P_3}$ ~~P3~~)

An Order is a sequence of OrderItems where each OrderItem is one product. An Order could appear as:

Order
OrderItem 4x6 print
OrderItem coffee mug

Please amend the paragraph beginning on page 19, line 16 as follows:

Fig. 6 is a flowchart summarizing overall operation of the system. As shown at step 600, the bit vectors mapping product availability for each fulfiller are routinely updated from the database. This updating includes adding/deleting fulfillers to a vector of fulfillers. The updating may include refreshing each bit according to periodic updates in inventory tables, in cases of extended implementations of the preferred embodiment that incorporate inventory data. During the update, each fulfiller is allotted a bit vector with a matching number of elements as the product hash table has cells corresponding to all the products (P_1, P_2, ... P_N) provided by the middleman. A new vector is built for each fulfiller, wherein each element, or bit, in the vector (which corresponds to each product type of the middleman, in the product hash table) is set to 1~~,~~ (for true)~~,~~ if the fulfiller provides that product. Otherwise, that bit is set to 0~~,~~ (for false) if the fulfiller does not provide this type of product. Therefore, if the middleman offers three types of products, and a particular fulfiller ~~does~~ provides the second and third products (as represented by the product hash table) the fulfiller's bit vector would be set to "011".

Please amend the paragraph beginning on page 20, line 10 as follows:

At step 620, the order ~~scheduler~~ engine (described above as 520) fetches a fulfillment order request. The order parser had already populated the database with the order data when the XML-encoded order request was received by either an HTTP or an FTP communication with the order-generating client (e.g., order entry at a PC). At step 630, the order items requested in the fulfillment order are mapped to the product items in the middleman's product line, the product hash table. Using the same bit vector mapping for the order's bit vector as was used at step 610 (in creating each fulfiller's bit vector: order items whose product type matched one of the middleman's product types), at step 630, an order bit vector is populated for the current order. Using the same logic as was used at step 610, at step 630, a new order vector is built for the current order, wherein each element, or bit, in the vector (which corresponds to each product type of the middleman, in the product hash table) is set to 1, for true, if the order requests that product. Otherwise, that bit is set to 0, for false if the order does not request this type of product. Therefore, if the middleman offers three types of products, and a particular order requests the first and second products (as sequenced in the product hash table) the fulfiller's bit vector would be set to "110".

Please amend the paragraph beginning on page 20, line 25 as follows:

At step 640, the order scheduler 530 ~~430~~ walks through the fulfiller vector, executing a bitwise & ("AND") operation on the current fulfiller bit vector and the order bit vector. If the bitwise & operation on these two bit vectors results in a bit vector with the same bit sequence as the order bit vector, then step 640 successfully fulfills the entire order with a single fulfiller (the most optimal condition), exits, and jumps ahead to step 670. If a single fuller cannot fulfill an entire order, then the order fulfillment must be split across multiple fulfillers; the method proceeds to the next step. At step 650, the order scheduler 530 ~~430~~ walks through the sort-ordered fulfiller vector, traverses each fulfiller's product vector, and executes a bitwise & ("AND") operation on the current fulfiller bit vector and an order item bit vector for each order item in the order bit vector. While the order bit vector is a “bitwise OR” of all the order items, the order item bit vector has only 1 bit set. For example, if the order bit vector is represented by “101”, the order item bit vector for the third product would be “001”. The order item bit vector is a temporary structure used to determine where the order items should be placed in the three-dimensional data structure shown in Fig. 4A. In this step the order scheduler 530 ~~430~~ places order items with multiple fulfillers, rather than placing the whole order with a single fulfiller. The following Java snippet tests an order item with the types of products offered by the earliest selected fulfiller:

```
(AllFSortedForAscendingProver[0].bv & OrderItem.bv) == OrderItem.bv
```

If the OrderItem cannot be placed in AllFSortedForAscendingProver [0] then the order scheduler 530 ~~430~~ tries the next index, AllFSortedForAscendingProver [1] and so on until this order item is placed with a fulfiller. This iteration is run against every order item sequentially in a single pass through a sort-ordered list of fulfiller.

Please amend the paragraph beginning on page 21, line 26 as follows:

At step 660, as the order items among the split order are being placed with providing fulfillers, the order scheduler 530 ~~430~~ populates a subsequent dimension (VP_i, described above) for the two-dimensional matrix mapping fulfiller products with the middleman's product line. This third, or higher, dimension represents collections (vectors) of order requests for the same type of product offering, but with subtler typing information in the description of an order item. For example, using an e-commercial photographic printing service as a fictitious fulfiller or middleman, three types of products are offered: 4x6 prints, 8x10 prints, and 11x14 prints. The product type, 4x6 prints, is represented by a cell in the 2-dimensional hash tables matrix (as previously described). However, an order may list a 4x6 print as two separate order items: one 4x6 print might be of the owner's dog, whereas the subsequent 4x6 print might be of the owner's cat. In this example scenario, both 4x6 order items would be placed with the same fulfiller, who provides 4x6 prints, and each placement would be recorded in a separate element along the VP_i vector indexed at the 4x6 product type for the current fulfiller.

Please amend the paragraph beginning on page 24, line 15 as follows:

```
// Return a list of fulfillers ordered by those closest to this zipCode
public Vector byProximity (String zipCode) {
  int zoneOfZipCode;
  Vector vectorOfFulfillers = new Vector();
```

```
    Vector fulfillersTmp;
    int step = 1;
    boolean keepGoing;
    int i;

    // The first digit of a zip code is the "national area" of the country.
    // The areas are:
    //    0 Northeast
    //    1 NewYork
    //    2 MidAtlantic
    //    3 Southeast
    //    4 GreatLakes
    //    5 Midwest
    //    6 Plains
    //    7 Southwest
    //    8 Western
    //    9 Pacific
    // This information is not online and I derived it by looking at post office
    // maps. So the names may not be correct but it is close enough for postal work.

    try {
        zoneOfZipCode = Integer.parseInt(zipCode.substring(0, 1));
    } catch(Exception e) {
        return vectorOfFulfillers; // passed in a malformed zip code
    }
    fulfillersTmp = Fulfiller.getByZone(zoneOfZipCode);
    for (i = 0; i < fulfillersTmp.size(); i++)
       vectorOfFulfillers.addElement(fulfillersTmp.elementAt(i));

    while (true) {
      keepGoing = false;

      // we may get a zone in the middle of the country so we need to step
      // away 1 zone at a time to make sure that we get the fulfillers closest
      // to this zone

      if (zoneOfZipCode + step <= Fulfiller.LAST_ZIP_ZONE) {
        fulfillersTmp = Fulfiller.getByZone(zoneOfZipCode + step);
        for (i = 0; i < fulfillersTmp.size(); i++)
           vectorOfFulfillers.addElement(fulfillersTmp.elementAt(i));
        keepGoing = true;
      }

      if (zoneOfZipCode - step >= Fulfiller.FIRST_ZIP_ZONE) {
        fulfillersTmp = Fulfiller.getByZone(zoneOfZipCode - step);
        for (i = 0; i < fulfillersTmp.size(); i++)
           vectorOfFulfillers.addElement(fulfillersTmp.elementAt(i));
        keepGoing = true;
      }

      if (keepGoing == true)
        step++;
      else
        break;

    } // end while true

    return vectorOfFulfillers;
  }
```

~~64.~~

Please remove pages 27-36, as this text has been added in an external computer program listing appendix contained on a compact disc per 37 CFR §1.96(c).

Please add the following paragraph after the paragraph beginning on page 2, line 5:

A computer program listing appendix has been included on a single compact disc, with a file named LS0026-computer-program-listing-appendix.doc with a size of 76 KB and a creation date of 7/18/05. The file and its content have been incorporated by reference.